Part 1: Written Problems (50pts)

1Q: Exercise R5.6. What is an implicit parameter? How does it differ from an explicit parameter?
1A: implicit parameters are accessed through calling a method of an object which then gives you access to the parameters within the object, being the implicit parameters. Explicit parameters are specified within parentheses when calling a method.

2Q: Using the English Length class from Module 10 (Week 1 Lecture), write a method **isLessThan(const English_Length & L);** that returns **true** if the implicit parameter is a shorter length than the explicit parameter, and **false** otherwise. Turn in the method prototype and implementation. You do not need to submit the entire .h or .cpp files.
2A:

```
bool English_length::isLessThan(const English_length & L)
{
        if (L < L.totalInches)
        {
                return true;
        }       else {
                return false;
        }
}
```

3Q:Using the method you just implemented, write a function called **min()** (**not** part of the English_Length class) that takes two **English_Length** parameters and returns the value of whichever is smaller.
3A:

```
Int min(const English_length & L, const English_length & M)
        {
                if(L.isLessThan(M))
                {
                        Return L;
                } else {
                        Return M;
                }
        }
```

4Q:Use **enum** to create a new data type called **Birthstone** that can only take the values of the Modern Birthstones recognized by the Jewelers of America (which are, in order: Garnet, Amethyst, Aquamarine, Diamond, Emerald, Moonstone, Ruby, Peridot, Sapphire, Opal, Topaz, Turquoise).

4A: `enum class Birthstone {` Garnet, Amethyst, Aquamarine, Diamond, Emerald, Moonstone, Ruby, Peridot, Sapphire, Opal, Topaz, Turquoise `};`

5Q:Using the type created above, write a function with prototype

- **Birthstone nextStone(Birthstone stone);**
- that returns the next birthstone for a given stone. Pay attention to rollovers.

5A:

```
Birthstone nextStone(Birthstone stone)
{
        switch(stone)
        {
        Case “Garnet”:
                Return “Amethyst”;
        Case “Amethyst”:
                Return “Aquamarine”;
        Case “Aquamarine”:
                Return “Diamond”;
        Case “Diamond”:
                Returrn “Emerald”;
        Case “Emerald”:
                Return “Moonstone”;
        Case “Moonstone”:
                Return “Ruby”;
        Case “Ruby”:
                Return “Peridot”;
        Case “Peridot”:
                Return “Sapphire”;
        Case “Sapphire”:
                Return “Opal”;
        Case “Opal”:
                Return “Topaz”;
        Case “Topaz”:
                Return “Turquoise”;
        Case “Turquoise”:
                Return “Garnet”;
        }
}
```

Part 2: Programming Assignment (50pts)

Background

This is the first part of a sequence of assignments that deal with creation of a social network website, which allows people to register as *socialites* who can share favorite web links and join fan clubs. For this part, you will focus on the socialites.

Description

There are two parts to the programming assignment.

- **Create the interface and partial implementation of a Socialite class according to the class specification below. The interface and implementations must be in separate files. The implementation will be upgraded in the next assignment.**
- **Write a main program (separate file) to test the Socialite class. The main program must test all methods.**

Specification for Socialite class:

Attributes

- **last name (string containing the user's last name)**
- **first name (string containing the user's first name)**
- **userid (string containing a unique userid - although you will not do anything to enforce uniqueness)**
- **picture (string containing the name of an image file)**
- **website to share (string containing a URL)**
- **website description (string containing a brief description of the website)**

Methods

- **default constructor (set all string fields to the empty string)**
- ***set* methods for each attribute (mutators)**
- ***get* methods for each attribute (inspectors)**
- ***text output* - outputs values of attributes in text format. This method should take an ostream reference as input allowing it to write to either console or a file.**

- ***HTML output*** **- outputs values of attributes as an HTML file. (See attached examples.) This method should take an ostream reference as input allowing it to write to either console or a file.**

NOTE: the layout of the text output and the HTML output is up to you. However all the attributes should be output. For an example of an HTML output see TonyD.html. This is a good chance for you to show off your HTML chops!

Main Program

Have your program create 5 Socialites, using the default constructor and updating info with the various *set* mutators. One Socialite should be yourself, while the others should include an entertainer/sports figure, a political figure, a cartoon character and someone else.

For each Socialite, create a unique userid (your program doesn't have to do anything intelligent like verify this and/or assign it automatically) and supply a description and a valid Internet URL for a picture. (You do *not* have to use a real picture of yourself - but provide a picture.)

Use the text output and HTML output methods for each Socialite (for testing purposes you may just pass in the output stream, but for your submission you'll want to pass in file output streams).

System Manual:

Socialite Header File:

Public: constructors, mutators, inspectors.

Socialite(); :
This is used as the default constructor of the object.

Socialite(string lastName, string firstName, string userID, string picture, string website, string webdes); :
This can be used as an alternative constructor by including strings for the objects last name, first name, user ID, picture, website, and website description.

setLastName(string newName);
setFirstName(string newName);
setUserID(string newUserID);
setPicture(string newPicture);
setWebsite(string newWebsite);
setWebDes(string newDesc);

—
All of these are used to mutate an object's attributes, assigning the attribute named after "set" with the element passed to the method.

string getLastName() const;
string getFirstName() const;
string getUserID() const;
string getPicture() const;
string getWebsite() const;
string getWebDes() const;

__
These methods are used as inspectors to retrieve and show different attributes of the object it is used on. Returning the attribute that is named after “get” in the method name.

ostream & toString(ostream &out);
ostream & toHTML(ostream &out);

__
toString and toHTML are both meant to do the same thing but are implemented differently in the implementation file.

Private:

string lastName_;
string firstName_;
string userID_;
string picture_;
string website_;
string webdes_;

Each of these strings are meant to hold information for the differently named attributes. With picture and website meant to hold links to an image and a website.

Socialite Implementation File:

Socialite::Socialite():

Used as the default constructor and automatically sets the attributes values to “” or null.

Socialite::Socialite(string lastName, string firstName, string userID, string picture, string website, string webdes):

Used as an alternative constructor and automatically sets the attributes values to the passed elements.

setLastName(string newName);
setFirstName(string newName);
setUserID(string newUserID);
setPicture(string newPicture);
setWebsite(string newWebsite);
setWebDes(string newDesc);

—

All of these are used to mutate an object’s attributes, assigning the attribute named after “set” with the element passed to the method all the same way.

string getLastName() const;
string getFirstName() const;
string getUserID() const;
string getPicture() const;

string getWebsite() const;
string getWebDes() const;

—
These methods are used as inspectors to retrieve and show different attributes of the object it is used on. Returning the attribute that is named after “get” in the method name all the same way.

ostream & toString(ostream &out);
ostream & toHTML(ostream &out);

—
toString and toHTML both export an object but are formatted differently.

toString exports the object's attributes in a reader friendly way as if to be sent to a text file.

toHTML exports the object’s attributes so that it can be written to an HTML file and understood by a browser to display the information so that it is user friendly.

Main():

—

First all of the files needed to export all of the profiles are assigned, totalling 10 files in all. The txt. Files are named “textOut” and then a number on the end (1-5), the html. Files are named “Site” and then a number on the end (1-5).

Then the object for my own profile is created by declaring a new “Socialite” object named “Reed”. The attributes are assigned appropriate values including links for an image and a website along with a string for a website description. I then call both the “toString” method and the “toHTML” method to export the object’s information to a txt file and an HTML file.

These steps are then repeated 4 more times for the other people needed for this project.